```
From 9a1dd3e0c27d68e32450be5538b864c2b115ee15 Mon Sep 17 00:00:00 2001
From: Karel Miko <karel.miko@gmail.com>
Date: Tue, 21 Apr 2026 20:14:12 +0200
Subject: [PATCH] fix GHSA-24c2-gp6c-24c6

---
 CryptX.xs                     |  23 ++++
 inc/CryptX_PK_DH.xs.inc       |   4 +
 inc/CryptX_PK_DSA.xs.inc      |   7 +-
 inc/CryptX_PK_ECC.xs.inc      |   7 +-
 inc/CryptX_PK_Ed25519.xs.inc |   2 +
 inc/CryptX_PK_RSA.xs.inc      |   6 +
 inc/CryptX_PK_X25519.xs.inc  |   2 +
 t/GHSA-24c2-gp6c-24c6.t       | 259 ++++++++++++++++++++++++++++++++++
 8 files changed, 308 insertions(+), 2 deletions(-)
 create mode 100644 t/GHSA-24c2-gp6c-24c6.t

diff --git a/CryptX.xs b/CryptX.xs
index 55d2613dc..9dea53e2e 100644
--- a/CryptX.xs
+++ b/CryptX.xs
@@ -130,30 +130,35 @@ typedef struct prng_struct {            /* used by Crypt::PRNG */
 typedef struct rsa_struct {             /* used by Crypt::PK::RSA */
   prng_state pstate;
   int pindex;
+  IV last_pid;
   rsa_key key;
 } *Crypt__PK__RSA;

 typedef struct dsa_struct {             /* used by Crypt::PK::DSA */
   prng_state pstate;
   int pindex;
+  IV last_pid;
   dsa_key key;
 } *Crypt__PK__DSA;

 typedef struct dh_struct {              /* used by Crypt::PK::DH */
   prng_state pstate;
   int pindex;
+  IV last_pid;
   dh_key key;
 } *Crypt__PK__DH;

 typedef struct ecc_struct {             /* used by Crypt::PK::ECC */
   prng_state pstate;
   int pindex;
+  IV last_pid;
   ecc_key key;
 } *Crypt__PK__ECC;

 typedef struct ed25519_struct {         /* used by Crypt::PK::Ed25519 */
   prng_state pstate;
   int pindex;
+  IV last_pid;
   curve25519_key key;
   int initialized;
 } *Crypt__PK__Ed25519;
@@ -161,6 +166,7 @@ typedef struct ed25519_struct {         /* used by Crypt::PK::Ed25519 */
 typedef struct x25519_struct {          /* used by Crypt::PK::X25519 */
   prng_state pstate;
   int pindex;
+  IV last_pid;
   curve25519_key key;
   int initialized;
```

```
 } *Crypt__PK__X25519;
@@ -197,6 +203,23 @@ STATIC int cryptx_internal_password_cb_getpw(void **p, unsigned long
*l, void *u
   return 0;
 }

+STATIC void cryptx_internal_pk_prng_reseed(prng_state *state, int pindex, IV *last_pid) {
+  IV curpid = (IV)PerlProc_getpid();
+  unsigned char entropy_buf[40];
+  int rv;
+
+  if (*last_pid == curpid) return;
+
+  if (rng_get_bytes(entropy_buf, sizeof(entropy_buf), NULL) != sizeof(entropy_buf)) {
+    croak("FATAL: rng_get_bytes failed");
+  }
+  rv = prng_descriptor[pindex].add_entropy(entropy_buf, sizeof(entropy_buf), state);
+  if (rv != CRYPT_OK) croak("FATAL: PRNG_add_entropy failed: %s", error_to_string(rv));
+  rv = prng_descriptor[pindex].ready(state);
+  if (rv != CRYPT_OK) croak("FATAL: PRNG_ready failed: %s", error_to_string(rv));
+  *last_pid = curpid;
+}
+
 STATIC void cryptx_internal_password_cb_free(void *p) {
   dTHX; /* fetch context */
   Safefree(p);
diff --git a/inc/CryptX_PK_DH.xs.inc b/inc/CryptX_PK_DH.xs.inc
index c6b845610..38cf81424 100644
--- a/inc/CryptX_PK_DH.xs.inc
+++ b/inc/CryptX_PK_DH.xs.inc
@@ -11,6 +11,7 @@ _new(Class)
         if (!RETVAL) croak("FATAL: Newz failed");
         RETVAL->key.type = -1;
         RETVAL->pindex = find_prng("chacha20");
+        RETVAL->last_pid = (IV)PerlProc_getpid();
         if (RETVAL->pindex == -1) {
           Safefree(RETVAL);
           croak("FATAL: find_prng('chacha20') failed");
@@ -29,6 +30,7 @@ _generate_key_size(Crypt::PK::DH self, int groupsize=256)
     PPCODE:
     {
         int rv;
+        cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
         rv = dh_set_pg_groupsize(groupsize, &self->key);
         if (rv != CRYPT_OK) croak("FATAL: dh_set_pg_groupsize failed: %s",
error_to_string(rv));
        rv = dh_generate_key(&self->pstate, self->pindex, &self->key);
@@ -43,6 +45,7 @@ _generate_key_gp(Crypt::PK::DH self, char *g, char *p)
        int rv;
        unsigned char pbin[1024], gbin[512];
        unsigned long plen=sizeof(pbin), glen=sizeof(gbin);
+       cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);

        if (p && strlen(p) > 0 && g && strlen(g) > 0) {
          rv = radix_to_bin(p, 16, pbin, &plen);
@@ -66,6 +69,7 @@ _generate_key_dhparam(Crypt::PK::DH self, SV * dhparam)
        int rv;
        unsigned char *data=NULL;
        STRLEN data_len=0;
+       cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
        data = (unsigned char *)SvPVbyte(dhparam, data_len);
        /* load d p q */
        rv = dh_set_pg_dhparam(data, (unsigned long)data_len, &self->key);
diff --git a/inc/CryptX_PK_DSA.xs.inc b/inc/CryptX_PK_DSA.xs.inc
index 1318a0d8c..128e2c4fa 100644
```

```
--- a/inc/CryptX_PK_DSA.xs.inc
+++ b/inc/CryptX_PK_DSA.xs.inc
@@ -11,6 +11,7 @@ _new(Class)
         if (!RETVAL) croak("FATAL: Newz failed");
         RETVAL->key.type = -1;
         RETVAL->pindex = find_prng("chacha20");
+        RETVAL->last_pid = (IV)PerlProc_getpid();
         if (RETVAL->pindex == -1) {
           Safefree(RETVAL);
           croak("FATAL: find_prng('chacha20') failed");
@@ -29,6 +30,7 @@ _generate_key_size(Crypt::PK::DSA self, int group_size=30, int
modulus_size=256)
     PPCODE:
     {
         int rv;
+        cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
         /* gen the key */
         rv = dsa_make_key(&self->pstate, self->pindex, group_size, modulus_size, &self-
>key);
         if (rv != CRYPT_OK) croak("FATAL: dsa_make_key failed: %s", error_to_string(rv));
@@ -42,6 +44,7 @@ _generate_key_dsaparam(Crypt::PK::DSA self, SV * dsaparam)
         int rv;
         unsigned char *data=NULL;
         STRLEN data_len=0;
+        cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
         data = (unsigned char *)SvPVbyte(dsaparam, data_len);
         /* load d p q */
         rv = dsa_set_pqg_dsaparam(data, (unsigned long)data_len, &self->key);
@@ -59,6 +62,7 @@ _generate_key_pqg_hex(Crypt::PK::DSA self, char *p, char *q, char *g)
         int rv;
         unsigned char pbin[512], qbin[512], gbin[512];
         unsigned long plen=sizeof(pbin), qlen=sizeof(qbin), glen=sizeof(gbin);
+        cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
         if (!p || !strlen(p) || !q || !strlen(q) || !g || !strlen(g)) {
           croak("FATAL: generate_key_pqg_hex incomplete args");
         }
@@ -342,6 +346,7 @@ encrypt(Crypt::PK::DSA self, SV * data, const char * hash_name = "SHA1")
         unsigned long buffer_len = 1024;

         data_ptr = (unsigned char *)SvPVbyte(data, data_len);
+        cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);

         hash_id = cryptx_internal_find_hash(hash_name);
         if (hash_id == -1) croak("FATAL: find_hash failed for '%s'", hash_name);
@@ -385,6 +390,7 @@ sign_hash(Crypt::PK::DSA self, SV * data, const char * hash_name =
"SHA1")
         STRLEN data_len = 0;

         data_ptr = (unsigned char *)SvPVbyte(data, data_len);
+        cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
         if (ix == 1) {
           id = cryptx_internal_find_hash(hash_name);
           if (id == -1) croak("FATAL: find_hash failed for '%s'", hash_name);
@@ -436,4 +442,3 @@ DESTROY(Crypt::PK::DSA self)
     CODE:
         if (self->key.type != -1) { dsa_free(&self->key); self->key.type = -1; }
         Safefree(self);
-
diff --git a/inc/CryptX_PK_ECC.xs.inc b/inc/CryptX_PK_ECC.xs.inc
index cc79796f6..8a83d04a3 100644
--- a/inc/CryptX_PK_ECC.xs.inc
+++ b/inc/CryptX_PK_ECC.xs.inc
@@ -10,6 +10,7 @@ _new(Class)
         Newz(0, RETVAL, 1, struct ecc_struct);
         if (!RETVAL) croak("FATAL: Newz failed");
```

```
          RETVAL->pindex = find_prng("chacha20");
  +       RETVAL->last_pid = (IV)PerlProc_getpid();
          RETVAL->key.type = -1;
          if (RETVAL->pindex == -1) {
            Safefree(RETVAL);
  @@ -29,6 +30,7 @@ generate_key(Crypt::PK::ECC self, SV *curve)
       PPCODE:
       {
          int rv;
  +       cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
          /* setup dp structure */
          rv = cryptx_internal_ecc_set_curve_from_SV(&self->key, curve); /* croaks on error
  */
          if (rv != CRYPT_OK) croak("FATAL: ecc_set_curve failed: %s", error_to_string(rv));
  @@ -377,6 +379,7 @@ encrypt(Crypt::PK::ECC self, SV * data, const char * hash_name = "SHA1")
          unsigned long buffer_len = 1024;

          data_ptr = (unsigned char *)SvPVbyte(data, data_len);
  +       cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);

          hash_id = cryptx_internal_find_hash(hash_name);
          if (hash_id == -1) croak("FATAL: find_hash failed for '%s'", hash_name);
  @@ -429,6 +432,7 @@ sign_hash(Crypt::PK::ECC self, SV * data, const char * hash_name = NULL,
  const c
          }

          data_ptr = (unsigned char *)SvPVbyte(data, data_len);
  +       cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
          if (ix == 1 || ix == 2) {
            /* for backward compatibility, if hash_name is NULL, use SHA1 */
            if (hash_name == NULL) {
  @@ -485,6 +489,7 @@ verify_hash(Crypt::PK::ECC self, SV * sig, SV * data, const char *
  hash_name = "

          data_ptr = (unsigned char *)SvPVbyte(data, data_len);
          sig_ptr = (unsigned char *)SvPVbyte(sig, sig_len);
  +       cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
          if (ix == 1 || ix == 2) {
            id = cryptx_internal_find_hash(hash_name);
            if (id == -1) croak("FATAL: find_hash failed for '%s'", hash_name);
  @@ -553,6 +558,7 @@ recovery_pub(Crypt::PK::ECC self, SV * sig, SV* hash, SV* recid = NULL)

          sig_ptr = (unsigned char *)SvPVbyte(sig, sig_len);
          hash_ptr = (unsigned char *)SvPVbyte(hash, hash_len);
  +       cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
          if (ix == 1) {
            sig_opts.type = LTC_ECCSIG_RFC7518;
          }
  @@ -574,4 +580,3 @@ DESTROY(Crypt::PK::ECC self)
       CODE:
          if (self->key.type != -1) { ecc_free(&self->key); self->key.type = -1; }
          Safefree(self);
  -
  diff --git a/inc/CryptX_PK_Ed25519.xs.inc b/inc/CryptX_PK_Ed25519.xs.inc
  index d50a84c03..fca73f90f 100644
  --- a/inc/CryptX_PK_Ed25519.xs.inc
  +++ b/inc/CryptX_PK_Ed25519.xs.inc
  @@ -11,6 +11,7 @@ _new(Class)
          if (!RETVAL) croak("FATAL: Newz failed");
          RETVAL->initialized = 0;
          RETVAL->pindex = find_prng("chacha20");
  +       RETVAL->last_pid = (IV)PerlProc_getpid();
          if (RETVAL->pindex == -1) {
            Safefree(RETVAL);
            croak("FATAL: find_prng('chacha20') failed");
```

```
@@ -30,6 +31,7 @@ generate_key(Crypt::PK::Ed25519 self)
     {
         int rv;
         self->initialized = 0;
+        cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
         rv = ed25519_make_key(&self->pstate, self->pindex, &self->key);
         if (rv != CRYPT_OK) croak("FATAL: ed25519_make_key failed: %s",
error_to_string(rv));
         self->initialized = 1;
diff --git a/inc/CryptX_PK_RSA.xs.inc b/inc/CryptX_PK_RSA.xs.inc
index 9f4b78f94..a425e367f 100644
--- a/inc/CryptX_PK_RSA.xs.inc
+++ b/inc/CryptX_PK_RSA.xs.inc
@@ -11,6 +11,7 @@ _new(Class)
         if (!RETVAL) croak("FATAL: Newz failed");
         RETVAL->key.type = -1;
         RETVAL->pindex = find_prng("chacha20");
+        RETVAL->last_pid = (IV)PerlProc_getpid();
         if (RETVAL->pindex == -1) {
           Safefree(RETVAL);
           croak("FATAL: find_prng('chacha20') failed");
@@ -30,6 +31,7 @@ generate_key(Crypt::PK::RSA self, int key_size=256, long key_e=65537)
     {
         /* key_size is in octets */
         int rv;
+        cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
         /* gen the key */
         rv = rsa_make_key(&self->pstate, self->pindex, key_size, key_e, &self->key);
         if (rv != CRYPT_OK) croak("FATAL: rsa_make_key failed: %s", error_to_string(rv));
@@ -362,6 +364,7 @@ encrypt(Crypt::PK::RSA self, SV * data, const char * padding = "oaep",
const cha
         data_ptr = (unsigned char *)SvPVbyte(data, data_len);

         RETVAL = newSVpvn(NULL, 0); /* undef */
+        cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
         if (strnEQ(padding, "oaep", 4)) {
           mgf_hash_id = cryptx_internal_find_hash(mgf_hash);
           if (mgf_hash_id == -1) croak("FATAL: find_hash failed for '%s'", mgf_hash);
@@ -427,6 +430,7 @@ decrypt(Crypt::PK::RSA self, SV * data, const char * padding = "oaep",
const cha
         data_ptr = (unsigned char *)SvPVbyte(data, data_len);

         RETVAL = newSVpvn(NULL, 0); /* undef */
+        cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
         if (strnEQ(padding, "oaep", 4)) {
           mgf_hash_id = cryptx_internal_find_hash(mgf_hash);
           if (mgf_hash_id == -1) croak("FATAL: find_hash failed for '%s'", mgf_hash);
@@ -491,6 +495,7 @@ sign_hash(Crypt::PK::RSA self, SV * data, const char * hash_name =
"SHA1", const
         ltc_rsa_op_parameters rsa_oparams;

         data_ptr = (unsigned char *)SvPVbyte(data, data_len);
+        cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
         if (ix == 1) {
           hash_id = cryptx_internal_find_hash(hash_name);
           if (hash_id == -1) croak("FATAL: find_hash failed for '%s'", hash_name);
@@ -561,6 +566,7 @@ verify_hash(Crypt::PK::RSA self, SV * sig, SV * data, const char *
hash_name = "

         data_ptr = (unsigned char *)SvPVbyte(data, data_len);
         sig_ptr = (unsigned char *)SvPVbyte(sig, sig_len);
+        cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
         if (ix == 1) {
           hash_id = cryptx_internal_find_hash(hash_name);
           if (hash_id == -1) croak("FATAL: find_hash failed for '%s'", hash_name);
```

```
diff --git a/inc/CryptX_PK_X25519.xs.inc b/inc/CryptX_PK_X25519.xs.inc
index d5c168ac2..f5792f251 100644
--- a/inc/CryptX_PK_X25519.xs.inc
+++ b/inc/CryptX_PK_X25519.xs.inc
@@ -11,6 +11,7 @@ _new(Class)
         if (!RETVAL) croak("FATAL: Newz failed");
         RETVAL->initialized = 0;
         RETVAL->pindex = find_prng("chacha20");
+        RETVAL->last_pid = (IV)PerlProc_getpid();
         if (RETVAL->pindex == -1) {
           Safefree(RETVAL);
           croak("FATAL: find_prng('chacha20') failed");
@@ -30,6 +31,7 @@ generate_key(Crypt::PK::X25519 self)
     {
         int rv;
         self->initialized = 0;
+        cryptx_internal_pk_prng_reseed(&self->pstate, self->pindex, &self->last_pid);
         rv = x25519_make_key(&self->pstate, self->pindex, &self->key);
         if (rv != CRYPT_OK) croak("FATAL: x25519_make_key failed: %s",
error_to_string(rv));
         self->initialized = 1;
diff --git a/t/GHSA-24c2-gp6c-24c6.t b/t/GHSA-24c2-gp6c-24c6.t
new file mode 100644
index 000000000..543899224
--- /dev/null
+++ b/t/GHSA-24c2-gp6c-24c6.t
@@ -0,0 +1,259 @@
+use strict;
+use warnings;
+use Config;
+use POSIX ();
+use Test::More;
+
+plan skip_all => "fork not available on this platform" unless $Config{d_fork};
+
+use IO::Handle;
+use Crypt::PK::DH;
+use Crypt::PK::DSA;
+use Crypt::PK::ECC;
+use Crypt::PK::Ed25519;
+use Crypt::PK::RSA;
+use Crypt::PK::X25519;
+use Crypt::PRNG;
+use Crypt::PRNG::ChaCha20;
+use Crypt::PRNG::Fortuna;
+use Crypt::PRNG::RC4;
+use Crypt::PRNG::Sober128;
+use Crypt::PRNG::Yarrow;
+
+sub fork_capture {
+  my ($code) = @_;
+
+  pipe(my $child_read, my $child_write) or BAIL_OUT("pipe failed: $!");
+  binmode $child_read;
+  binmode $child_write;
+  $child_write->autoflush(1);
+
+  my $pid = fork();
+  BAIL_OUT("fork failed: $!") unless defined $pid;
+
+  if ($pid == 0) {
+    my $ok = eval {
+      close $child_read or die "close child_read failed: $!";
+      my $value = $code->();
+      die "callback returned undef" unless defined $value;
```

```
+      print {$child_write} $value;
+      close $child_write or die "close child_write failed: $!";
+      1;
+    };
+    POSIX::_exit($ok ? 0 : 1);
+  }
+
+  close $child_write or BAIL_OUT("close child_write failed: $!");
+
+  my $parent_value = eval { $code->() };
+  my $parent_error = $@;
+  waitpid($pid, 0) if $parent_error;
+  BAIL_OUT("parent callback failed: $parent_error") if $parent_error;
+
+  my $child_value = do { local $/; <$child_read> };
+  close $child_read or BAIL_OUT("close child_read failed: $!");
+
+  waitpid($pid, 0);
+  return ($parent_value, $child_value, $? >> 8);
+}
+
+sub expect_fork_divergence {
+  my ($name, $code) = @_;
+
+  subtest $name => sub {
+    my ($parent_value, $child_value, $child_status) = fork_capture($code);
+
+    ok(defined $parent_value && length $parent_value, 'parent produced output');
+    ok(defined $child_value && length $child_value, 'child produced output');
+    is($child_status, 0, 'child exited cleanly');
+    isnt($parent_value, $child_value, 'parent and child diverge after fork');
+  };
+}
+
+expect_fork_divergence(
+  'Crypt::PRNG bytes',
+  do {
+    my $prng = Crypt::PRNG->new;
+    sub {
+      return $prng->bytes_hex(32);
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PRNG::Fortuna bytes',
+  do {
+    my $prng = Crypt::PRNG::Fortuna->new;
+    sub {
+      return $prng->bytes_hex(32);
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PRNG::Yarrow bytes',
+  do {
+    my $prng = Crypt::PRNG::Yarrow->new;
+    sub {
+      return $prng->bytes_hex(32);
+    };
+  },
+);
+
+expect_fork_divergence(
```

```
+  'Crypt::PRNG::RC4 bytes',
+  do {
+    my $prng = Crypt::PRNG::RC4->new;
+    sub {
+      return $prng->bytes_hex(32);
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PRNG::Sober128 bytes',
+  do {
+    my $prng = Crypt::PRNG::Sober128->new;
+    sub {
+      return $prng->bytes_hex(32);
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PRNG::ChaCha20 bytes',
+  do {
+    my $prng = Crypt::PRNG::ChaCha20->new;
+    sub {
+      return $prng->bytes_hex(32);
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PK::RSA generate_key',
+  do {
+    my $pk = Crypt::PK::RSA->new;
+    sub {
+      $pk->generate_key(128, 65537);
+      return $pk->export_key_jwk_thumbprint('SHA256');
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PK::ECC generate_key',
+  do {
+    my $pk = Crypt::PK::ECC->new;
+    sub {
+      $pk->generate_key('secp256k1');
+      return $pk->export_key_jwk_thumbprint('SHA256');
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PK::DH generate_key',
+  do {
+    my $pk = Crypt::PK::DH->new;
+    sub {
+      $pk->generate_key(128);
+      return $pk->key2hash->{y};
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PK::DSA generate_key',
+  do {
```

```
+    my $pk = Crypt::PK::DSA->new;
+    sub {
+      $pk->generate_key(20, 128);
+      return $pk->key2hash->{y};
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PK::Ed25519 generate_key',
+  do {
+    my $pk = Crypt::PK::Ed25519->new;
+    sub {
+      $pk->generate_key;
+      return $pk->key2hash->{pub};
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PK::X25519 generate_key',
+  do {
+    my $pk = Crypt::PK::X25519->new;
+    sub {
+      $pk->generate_key;
+      return $pk->key2hash->{pub};
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PK::RSA encrypt',
+  do {
+    my $pk = Crypt::PK::RSA->new('t/data/cryptx_pub_rsa1.der');
+    sub {
+      return unpack 'H*', $pk->encrypt('secret message', 'oaep', 'SHA256');
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PK::RSA sign_message',
+  do {
+    my $pk = Crypt::PK::RSA->new('t/data/cryptx_priv_rsa1.der');
+    sub {
+      return unpack 'H*', $pk->sign_message('secret message', 'SHA256', 'pss', 12);
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PK::ECC encrypt',
+  do {
+    my $pk = Crypt::PK::ECC->new('t/data/cryptx_pub_ecc1.der');
+    sub {
+      return unpack 'H*', $pk->encrypt('secret message');
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PK::ECC sign_message',
+  do {
+    my $pk = Crypt::PK::ECC->new('t/data/cryptx_priv_ecc1.der');
+    sub {
```

```
+      return unpack 'H*', $pk->sign_message('secret message', 'SHA256');
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PK::DSA encrypt',
+  do {
+    my $pk = Crypt::PK::DSA->new('t/data/cryptx_pub_dsa1.der');
+    sub {
+      return unpack 'H*', $pk->encrypt('secret message');
+    };
+  },
+);
+
+expect_fork_divergence(
+  'Crypt::PK::DSA sign_message',
+  do {
+    my $pk = Crypt::PK::DSA->new('t/data/cryptx_priv_dsa1.der');
+    sub {
+      return unpack 'H*', $pk->sign_message('secret message', 'SHA256');
+    };
+  },
+);
+
+done_testing;
```